

# Linguagem de Programação

## Arquivos em C e Parâmetros do Programa

Alexandre Mello

Fatec Campinas

2023

# Roteiro

## 1 Arquivos

- Introdução a Arquivos em C
- Nomes e Extensões
- Tipos de Arquivos
- Caminhos Absolutos e Relativos

## 2 Arquivos Textos

- Ponteiro para Arquivos
- Abrindo um Arquivo
- Lendo um Arquivo
- Escrevendo em um Arquivo

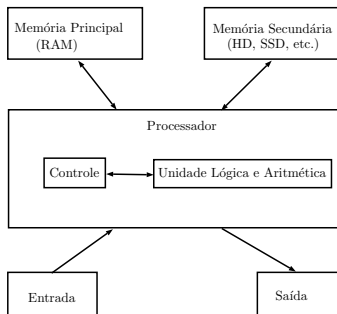
## 3 Exemplos

## 4 Parâmetros do programa: argc e argv

## 5 Informações Extras: fscanf para ler **int**, **double**, etc.

# Tipos de Memória

- Quando vimos a organização básica de um sistema computacional, havia somente um tipo de memória.
- Mas na maioria dos sistemas, a memória é dividida em dois tipos:



# Tipos de Memória

- A memória principal (Random Access Memory) utilizada na maioria dos computadores, usa uma tecnologia que requer alimentação constante de energia para que informações sejam preservadas.



# Tipos de Memória

- A memória secundária (como Hard Disks ou SSD) utilizada na maioria dos computadores, usa uma outra tecnologia que NÃO requer alimentação constante de energia para que informações sejam preservadas.



# Tipos de Memória

- Todos os programas executam na RAM, e por isso quando o programa termina ou acaba energia, as informações do programa são perdidas.
- Para podermos gravar informações de forma *persistente*, devemos escrever estas informações em arquivos na memória secundária.
- A memória secundária possui algumas características:
  - ▶ É muito mais lenta que a RAM.
  - ▶ É muito mais barata que a memória RAM.
  - ▶ Possui maior capacidade de armazenamento.
- Sempre que nos referirmos a um arquivo, estamos falando de informações armazenadas em memória secundária.

# Nomes e Extensões

- Arquivos são identificados por um nome.
- O nome de um arquivo pode conter uma extensão que indica o conteúdo do arquivo.

## Algumas extensões

arq.txt	arquivo texto simples
arq.c	código fonte em C
arq.pdf	<i>portable document format</i>
arq.html	arquivo para páginas WWW ( <i>hypertext markup language</i> )
arq*	arquivo executável (UNIX)
arq.exe	arquivo executável (Windows)

# Tipos de arquivos

Arquivos podem ter o mais variado conteúdo, mas do ponto de vista dos programas existem apenas dois tipos de arquivo:

**Arquivo texto:** Armazena caracteres que podem ser mostrados diretamente na tela ou modificados por um editor de textos simples. Exemplos: código fonte C, documento texto simples, páginas HTML.

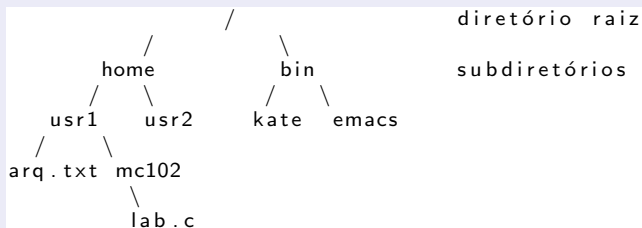
**Arquivo binário:** Sequência de bits sujeita às convenções dos programas que o gerou, não legíveis diretamente. Exemplos: arquivos executáveis, arquivos compactados, documentos do Word.



# Diretório

- Também chamado de pasta.
- Contém arquivos e/ou outros diretórios.

## Uma hierarquia de diretórios



# Caminhos Absolutos e Relativos

- O nome de um arquivo pode conter o seu diretório, ou seja, o caminho para encontrar este arquivo a partir da raiz.
- Desta forma o acesso a um arquivo pode ser especificado de duas formas:

**Caminho absoluto:** descrição de um caminho desde o diretório raiz.

```
/bin/emacs  
/home/usr1/arq.txt
```

**Caminho relativo:** descrição de um caminho a partir do diretório corrente.

```
arq.txt  
mc102/lab.c
```

## Arquivos texto em C

- Em C, para se trabalhar com arquivos devemos criar um ponteiro especial: **um ponteiro para arquivos**.

```
FILE *nome_variavel;
```

- O comando acima cria um ponteiro para arquivos, cujo nome da variável é o nome especificado.
- Após ser criado um ponteiro para arquivo, podemos associá-lo com um arquivo real do computador usando a função **fopen**.

```
FILE *arq1;  
arq1 = fopen("teste.txt", "r");
```

- Neste exemplo a variável ponteiro **arq1** fica apontando para o arquivo **teste.txt**.

# Arquivos texto em C

```
FILE *arq1;  
arq1 = fopen("teste.txt", "r");
```

- O primeiro parâmetro para **fopen** é uma string com o nome do arquivo
  - ▶ Pode ser absoluto, por exemplo: `"/user/eduardo/teste.txt"`
  - ▶ Pode ser relativo como no exemplo acima: `"teste.txt"`
- O segundo parâmetro é uma string informando como o arquivo será aberto.
  - ▶ Se para leitura ou gravação de dados, ou ambos.
  - ▶ Se é texto ou se é binário.
  - ▶ No nosso exemplo, o **"r"** significa que abrimos um arquivo texto para leitura.

# Abrindo um Arquivo Texto para Leitura

- Antes de acessar um arquivo, devemos abri-lo com a função `fopen()`.
- A função retorna um ponteiro para o arquivo em caso de sucesso, e em caso de erro a função retorna `NULL`.

## Abrindo o arquivo teste.txt

```
FILE *arq = fopen("teste.txt", "r");  
if (arq == NULL)  
    printf("Erro ao tentar abrir o arquivo teste.txt.");  
else  
    printf("Arquivo aberto para leitura.\n");
```

# Lendo Dados de um Arquivo Texto

- Para ler dados do arquivo aberto, usamos a função **fscanf()**, que é semelhante à função **scanf()**.
  - ▶ **int fscanf(ponteiro para arquivo, string de formato, variáveis).**
  - ▶ A única diferença para o **scanf**, é que devemos passar como primeiro parâmetro um ponteiro para o arquivo de onde será feita a leitura.

## Lendo dados do arquivo teste.txt

```
char aux;  
FILE *f = fopen ("teste.txt", "r");  
fscanf(f, "%c", &aux);  
printf("%c", aux);
```

# Lendo Dados de um Arquivo Texto

- Quando um arquivo é aberto, um **indicador de posição** no arquivo é criado, e este recebe a posição do início do arquivo.
- Para cada dado lido do arquivo, este indicador de posição é automaticamente incrementado para o próximo dado não lido.
- Eventualmente o indicador de posição chega ao fim do arquivo:
  - ▶ A função **fscanf** devolve um valor especial, **EOF** (*End Of File*), caso tente-se ler dados e o indicador de posição está no fim do arquivo.

# Lendo Dados de um Arquivo Texto

- Para ler todos os dados de um arquivo texto, basta usarmos um laço que será executado enquanto não chegarmos no fim do arquivo:

## Lendo dados do arquivo teste.txt

```
char aux;  
FILE *f = fopen ("teste.txt", "r");  
while (fscanf(f, "%c", &aux) != EOF)  
    printf("%c", aux);  
fclose(f);
```

- O comando **fclose** (no fim do código) deve sempre ser usado para fechar um arquivo que foi aberto.
  - ▶ Quando escrevemos dados em um arquivo, este comando garante que os dados serão efetivamente escritos no arquivo.



Exemplo de programa que imprime o conteúdo de um arquivo texto na tela:

```
#include <stdio.h>
```

```
int main() {  
    FILE *arq;  
    char aux, nomeArq[100];  
  
    printf("Entre com nome do arquivo:");  
    scanf("%s", nomeArq);  
    arq = fopen(nomeArq, "r");  
    if (arq == NULL)  
        printf("Erro ao abrir o arquivo");  
    else{  
        printf("——— Dados do arquivo:\n\n");  
        while(fscanf(arq,"%c",&aux) != EOF){  
            printf("%c",aux);  
        }  
    }  
    fclose(arq);  
}
```

# Lendo Dados de um Arquivo Texto

- Notem que ao realizar a leitura de um caractere, automaticamente o indicador de posição do arquivo se move para o próximo caractere.
- Ao chegar no fim do arquivo a função **fscanf** retorna o valor especial **EOF**.
- Para voltar ao início do arquivo você pode fechá-lo e abrí-lo mais uma vez, ou usar o comando **rewind**.

```
while ( fscanf (arq , "%c" , &aux) != EOF) {  
    printf ("%c" , aux );  
}  
  
printf {"\n\n ———Imprimindo novamente\n\n"} ;  
rewind (arq) ;  
  
while ( fscanf (arq , "%c" , &aux) != EOF) {  
    printf ("%c" , aux );  
}
```

# Escrevendo Dados em um Arquivo Texto

- Para escrever em um arquivo, ele deve ser aberto de forma apropriada, usando a opção **w**.
- Usamos a função **fprintf()**, semelhante à função **printf()**.
  - ▶ **int fprintf(ponteiro para arquivo, string de saída, variáveis)**
  - ▶ É semelhante ao **printf** mas notem que precisamos passar o ponteiro para o arquivo onde os dados serão escritos.

## Copiando dois arquivos

```
FILE *fr = fopen (" teste.txt", "r" ); //Abre arq. para leitura
FILE *fw = fopen (" saida.txt", "w" ); //Abre arq. para escrita
while ( fscanf(fr, "%c", &c) != EOF)
    fprintf(fw,"%c", c);
fclose(fr);
fclose(fw);
```

# Escrevendo Dados em um Arquivo Texto

Exemplo de programa que faz copia de um arquivo texto.

```
int main() {
    FILE *arqIn, *arqOut;
    char aux, nomeArqIn[100], nomeArqOut[100];

    printf("Entre com nome do arquivo de entrada:");
    scanf("%s", nomeArqIn);
    arqIn = fopen(nomeArqIn, "r");
    if (arqIn == NULL){
        printf("Erro ao abrir o arquivo: %s\n", nomeArqIn); return 1;
    }

    printf("Entre com nome do arquivo de saida:");
    scanf("%s", nomeArqOut);
    arqOut = fopen(nomeArqOut, "w");
    if (arqOut == NULL){
        printf("Erro ao abrir o arquivo: %s\n", nomeArqOut); return 1;
    }

    while(fscanf(arqIn, "%c", &aux) != EOF){
        fprintf(arqOut, "%c", aux);
    }

    fclose(arqIn);
    fclose(arqOut);
}
```

# fopen

Um pouco mais sobre a função **fopen()**.

```
FILE* fopen(const char *caminho, char *modo);
```

## Modos de abertura de arquivo texto

modo	operações	indicador de posição começa
r	leitura	início do arquivo
r+	leitura e escrita	início do arquivo
w	escrita	início do arquivo
w+	escrita e leitura	início do arquivo
a	(append) escrita	final do arquivo

# fopen

- Se um arquivo for aberto para leitura (**r**) e ele não existir, **fopen** devolve **NULL**.
- Se um arquivo for aberto para escrita ou escrita/leitura (**w** ou **w+**) e existir ele é apagado e criado;  
Se o arquivo não existir um novo arquivo é criado.
  - ▶ No modo **w** você poderá fazer apenas escritas e no modo **w+** você poderá fazer tanto escritas quanto leituras.
- Se um arquivo for aberto para leitura/escrita (**r+**) e existir ele **NÃO** é apagado;  
Se o arquivo não existir, **fopen** devolve **NULL**.

## Exemplo: Lendo um texto na memória

- Podemos ler todo o texto de um arquivo para um vetor (deve ser grande o suficiente!) e fazer qualquer alteração que julgarmos necessária.
- O texto alterado pode então ser sobrescrito sobre o texto anterior.
- Como exemplo, vamos fazer um programa que troca toda ocorrência da letra "a" por "A" em um texto.

# Lendo um texto na memória

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]){
    FILE *arq;
    char nomeArq[100];

    printf(" Digite o nome do arquivo: ");
    scanf("%s", nomeArq);
    arq = fopen(nomeArq, "r+");

    if(arq == NULL){
        printf(" Arquivo inexistente!\n");
        return 1;
    }

    //Vamos determinar o tamanho do arquivo
    char aux;
    int size=0;
    while(fscanf(arq, "%c", &aux) != EOF)
        size++;

    //Aloca-se string com espaco suficiente para o arquivo
    char *texto = malloc((size+1)*sizeof(char));
    .....
}
```



# Lendo um texto na memória

```
int main() {  
    .....  
    //Carrega o arquivo para a memória  
    rewind(arq);  
    int i=0;  
    while(fscanf(arq, "%c", &aux) != EOF){  
        texto[i] = aux;  
        i++;  
    }  
    texto[i] = '\0';  
    printf("%s", texto);    //Imprime texto original  
  
    //Escreve o arquivo modificado  
    rewind(arq);  
    i=0;  
    while(texto[i] != '\0'){  
        if(texto[i] == 'a')  
            texto[i] = 'A';  
        fprintf(arq, "%c", texto[i]);  
        i++;  
    }  
    free(texto);  
    fclose(arq);  
}
```

# Resumo para se Trabalhar com Arquivos

- Crie um ponteiro para arquivo: **FILE \*parq;**
- Abra o arquivo de modo apropriado associando-o a um ponteiro:
  - ▶ **parq = fopen(nomeArquivo, modo);** onde modo pode ser: **r, r+, w, w+**
- Leia dados do arquivo na memória.
  - ▶ **fscanf(parq, string-tipo-variavel, &variavel);**
  - ▶ Dados podem ser lidos enquanto **fscanf** não devolver **EOF**.
- Altere dados se necessário e escreva-os novamente em arquivo.
  - ▶ **fprintf(parq, string-tipo-variavel, variavel);**
- Todo arquivo aberto deve ser fechado.
  - ▶ **fclose(parq);**

# Argc e Argv

- Até então temos criado programas onde a função `main()` não tem parâmetros.
- Mas esta função pode receber dois parâmetros: `main(int argc, char *argv[])`.
  - ▶ `argc` (*argument counter*): indica o número de argumentos na linha de comando ao se executar o programa.
  - ▶ `*argv[]` (*argument vector*): é um vetor de ponteiros para caracteres (ou seja vetor de strings) que contém os argumentos da linha de comando, um em cada posição do vetor.

# Argc e Argv

O programa abaixo imprime cada um dos parâmetros passados na linha de comando:

```
#include <stdio.h>

int main(int argc, char *argv[]){
    int i;

    for(i=0; i<argc; i++){
        printf("%s\n", argv[i]);
    }
}
```

# Argc e Argv

- Seu uso é útil em programas onde dados de entrada são passados via linha de comando.
- Exemplo: dados a serem processados estão em um arquivo, cujo nome é passado na linha de comando.

```
//Este programa mostra o conteúdo de um arquivo texto cujo nome é passado como
//parâmetro do programa
int main(int argc, char *argv[]){
    int i;
    FILE *fp=NULL;

    fp = fopen(argv[1], "r+");
    if(fp == NULL){
        printf("Arquivo não existe!\n");
        return 1;
    }

    char aux;
    while(fscanf(fp, "%c", &aux) != EOF){
        printf("%c", aux);
    }
}
```

## Informações Extras: fscanf para **int**, **double**, etc.

- Você pode usar o **fscanf** como o **scanf** para ler dados em variáveis de outro tipo que não texto ou char.
  - ▶ Pode-se ler uma linha "1234" no arquivo texto para um **int** por exemplo:

```
int i;  
fscanf(arq, "%d", &i);
```

- O mesmo vale para o **fprintf** em relação ao **printf**.
  - ▶ Neste exemplo é escrito o texto "56" no arquivo.

```
int i=56;  
fprintf(arq, "%d", i);
```

- Você pode remover um arquivo usando a função **remove(string-nome-arq)**.